Vietnam J Comput Sci (2015) 2:57–66
DOI 10.1007/s40595-014-0031-8

REGULAR PAPER

A feasible MapReduce peer-to-peer framework for distributed computing applications

Ha Manh Tran · Synh Viet Uyen Ha · Tu Kha Huynh · Son Thanh Le

Received: 30 November 2013 / Accepted: 24 August 2014 / Published online: 16 September 2014

Abstract This article presents a MapReduce Peer-to-Peer (P2P) framework that enables a MapReduce implementation on P2P networks to support a class of MapReduce-based computing applications. This framework can be useful for researchers who cannot afford expensive and dedicated clusters for infrequent demands of solving distributed computing problems. The framework also allows Internet users from social and P2P network communities to perform large data processing experiments on distributed environment. The article describes the architecture and prototyping implementation of a MapReduce P2P system. The main features of this system include exploiting leisure computing resources on P2P networks efficiently for computation, providing MapReduce operations using task management for various distributed computing problems, and supporting peer failure management for improving fault tolerance on Internet environment. We have evaluated and compared the MapReduce P2P implementation with a Hadoop MapReduce implementation on local and global-scale networks. The article also includes a discussion of applying the framework to a realistic distributed case-based reasoning system.

Keywords MapReduce · Peer-to-peer network · Distributed computing · Hadoop

H. M. Tran (✉) · S. V. U. Ha · T. K. Huynh · S. T. Le
Computer Science and Engineering, International University-Vietnam National University, Ho Chi Minh City, Vietnam
e-mail: tmha@hcmiu.edu.vn

S. V. U. Ha
e-mail: hvusynh@hcmiu.edu.vn

T. K. Huynh
e-mail: hktu@hcmiu.edu.vn

S. T. Le
e-mail: ltson@hcmiu.edu.vn

1 Introduction

MapReduce [10] programming model combined with dedicated clusters of workstations enables distributed computing applications to process large amount of data in parallel and distribution with high performance. Solving distributed computing problems today thus becomes easier and more efficient using virtual clusters on high-performance computing clouds. However, users such as researchers or small groups of researchers who infrequently have demands of solving distributed problems cannot afford dedicated clusters and virtual clusters; or users from social and P2P network communities tend to exploit leisure computing resources shared among peers for distributed computing applications, such as distributed search engines on P2P networks, distributed online games on social networks. The applicability of the MapReduce model for this class of applications and users can be a demand.

Several P2P applications are successful in sharing and searching resources, such as sharing multimedia files [7, 15, 16, 22]. Other P2P applications concentrate on resource retrieval [4, 13, 17, 34] and distributed computing [25, 27, 31]. Peers in computing applications contribute computing resources, such as storage, bandwidth and processing power to solve tasks independently or collaboratively. P2P networks usually expose high degree of self-management, scalability and fault tolerance. Peers join and leave P2P networks without central administration. However, the applicability of P2P networks for distributed computing can be a challenge. Peers possess unstable and heterogeneous computing resources, while solving these distributed problems requires computing resources with some degree of stability and reliability. In addition, collaborating between peers over the Internet increases communication time, thus reducing the performance of solving these distributed problems.

We propose a MapReduce P2P framework for solving distributed computing problems. The aims of this framework are to exploit leisure resources on the P2P network rather than using dedicated clusters and provide a distributed computing environment for users who infrequently have demands of solving large distributed problems. The original framework published in the previous study [9] has basically applied MapReduce operations including *mapper* and *reducer* on a P2P network architecture. It features the capability of exploiting capable peers with sufficient storage, bandwidth and processing power on a P2P network for group establishment and data distribution. Based on the experiments of the previous study, we extend this framework to deal with peer failure, task management and performance evaluation in realistic distributed environment. The extensions of this framework in this article include:

- providing task and failure management components for the framework to control peer execution and failure
- modifying the Gnutella P2P protocol and supplementing these two components on the MapReduce P2P system
- evaluating MapReduce P2P prototyping implementation on both local and global-scale networks with a comparison to Hadoop MapReduce implementation [1].

The rest of the article is structured as follows: the next section provides the background of P2P networks and research activities related to applying the MapReduce model to P2P networks. Section 3 describes the architecture of the MapReduce P2P system and the extension of the Gnutella protocol. Section 4 presents the prototyping implementation of the system that includes the extended design of failure and task management components for peers. Performance evaluation contains several experiments reported in Sect. 5 with a comparison to the Hadoop MapReduce implementation. Section 6 concludes the article with a discussion of applying the framework to a realistic distributed case-based reasoning system.

2 Related work

P2P networks contain a large number of workstations that share storage, bandwidth and processor power to provide services. P2P networks possess the capability of maintaining the stability of the overlay networks where peers dynamically join and leave. P2P networks foster high scalability and fault tolerance, and reduce collaboration cost through ad hoc communication process. P2P networks are broadly classified by the level of network structure. A peer joins a P2P network by connecting to existing peers in an unstructured network or by connecting to well-defined peers based on an identifier in a structured network [5].

The structured P2P network is tightly controlled in topology: a peer joining this network is fixed in a logical location. This kind of networks uses distributed hash table (DHT) to generate uniquely consistent identifiers for both peers and resources. Peers keep resource indexes if they share the same identifier space, thus distributing fairly resources among peers and reducing the impact of peer failure. Peers maintain a list of the neighboring peers as a routing table. Search queries are forwarded to the neighboring peers which are closer to the resource indexes in the identifier space. The main disadvantages of these networks are the unbalanced load problem and unsupported fault tolerance. Several popular structured P2P systems are CAN [23], Chord [26], Pastry [24], Tapestry [36], Kademlia [20], etc. The unstructured P2P network is loosely controlled in topology: a peer joins this network by connecting to other peers in a random fashion. Peers maintain lists of resource indexes and the neighboring peers in their local repository. Search queries are flooded to the neighboring peers that in turn forward the queries to other peers in the network. Upon receiving the queries, peers return relevant resource indexes as queryhits. To avoid the traffic explosion of the flooding-based routing mechanism, query messages contain a time-to-live value that allows messages to reach a number of peers. The main disadvantages of these networks are the severe scalability problem and unstable success rate as the number of queries and peers considerably increases. Several popular unstructured P2P systems are Gnutella [14], Freenet [6], BitTorrent [7], etc.

The super peer P2P network is a hybrid network that combines the good characteristics of the client–server and P2P networks to deal with the problem of heterogeneous peers, i.e., incapable peers with limited storage, bandwidth and processing power cannot serve requests. The study of Yang et al. [35] has proposed a design of the super peer network that only considers capable peers as serving peers or super peers. The super peer network comprises several clusters connected to each other to form either structured or unstructured P2P networks. Each cluster based on the client–server architecture contains a super peer as a server and a set of peers as clients. Super peers forward queries and obtain queryhits on the super peer network, while clients send queries to, and receive queryhits from, their super peer. Each cluster can maintain several redundant super peers to deal with the single point of failure problem. The later version of the Gnutella protocol has included this super peer concept in the Gnutella P2P network. Ultrapeers or super peers form an unstructured network on top of the IP network. They represent peers to query and respond to the network, thus undertaking large amount of the network load, exploiting the inherent heterogeneity of the P2P network and tolerating the network faults.

The study of Fabrizio et al. [18,19] focuses on improving MapReduce implementations for distributed platforms such as Grid or P2P. In Internet-based computing environ-

ment, failures are likely to happen since peers join and leave the network at an unpredictable rate. The study deals with managing intermittent peer participation, master failure and job recovery issues of the MapReduce framework that can be applied to computational Grids or P2P systems. The study includes a proposal of a P2P MapReduce architecture, where each peer can act as either master or slave, thus creating a pool of backup masters. In case of the master failure, the backup master is promoted to the master by the election mechanism of the backup masters. Although the proposed system handles the master failure and job recovery, the system still suffers from the problem of heterogeneous peers. Peers are different in storage, bandwidth and computing power, thus choosing the master based on the smallest workload seems insufficient. When using this system for solving large distributed problems, the master failure activates processes of electing a new master and recovering several tasks. The system consumes a lot of time and resource for master election and task recovery management, thus reducing the performance of solving the problems. Moreover, when using the JXTA open source package to build a structured P2P network that possesses a controlled topology, peers cannot send group formation messages arbitrarily to other peers, the system thus encounters difficulty in choosing capable peers.

Our recent study [9] has proposed a framework of exploiting leisure resources including storage, bandwidth and processing power on peers to deal with the problem of peer heterogeneity, peer group formation, task assignment and data distribution for P2P computing applications. The framework employs the super peer P2P network where peers possess sufficiently processing capability for the basic MapReduce operations. The Gnutella protocol has been extended to establishing peer groups and task assignments. The framework also provides a mechanism of distributing datasets among peers. The evaluation of the MapReduce P2P prototyping system has exposed several issues: (i) distributing large datasets on peers can cause peer failures on high latency networks; (ii) different distributed problems require different *mapper* and *reducer* operations; and (iii) the prototyping implementation must be evaluated on large-scale networks. This study improves the framework by providing peer failure and task management components, implementing these components on the system and performing experiments on the Internet.

3 Architecture

The MapReduce P2P system architecture as shown in Fig. 1 contains peers and storage servers. A typical peer possesses network, group and MapReduce modules. The network module allows peers and modules to exchange information through different types of messages, e.g., a response of joining a group forwarded to the group module or a request

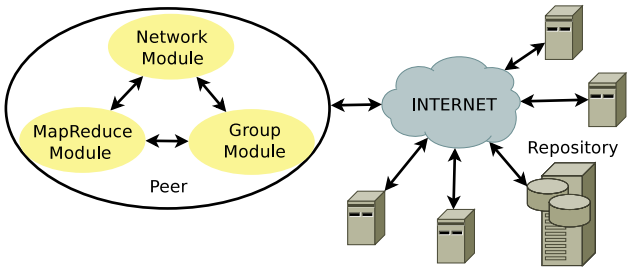

Fig. 1 MapReduce P2P system architecture

of checking peer alive forwarded to the neighboring peers. The group module manages several groups to which the peer belongs, acting as either the master or the slave. This module cooperates with other modules to maintain the stability of groups and provide the information of peers. The core MapReduce module controls the MapReduce operations of peers. This module involves generating and executing tasks, distributing and retrieving datasets for peers, and maintains local repository. In addition, the system requires some mediate repositories for the master and the slaves to upload and download the input datasets. Due to the symmetric characteristics of roles, peers are designed to possess the functions of both the master and the slave.

The MapReduce P2P system architecture possesses a super peer Gnutella network that contains super peers and peers. A super peer must satisfy some requirements, such as being not behind a firewall, having sufficient bandwidth, uptime and processing power. Super peers connect to each other to form a P2P network of super peers, while peers choose to connect to a super peer to form a cluster of the super peer and peers. Super peers act as proxies to the Gnutella P2P network for peers. Query mechanism uses client server paradigm in the cluster and P2P paradigm in the super peer network with various routing methods. The super peer network resolves heterogeneity problem, and also increases the scalability of the Gnutella network by reducing the number of incapable peers incorporating in query routing activities.

3.1 Gnutella protocol extension

The Gnutella protocol supports five types of messages: *ping* and *pong* used to probe the network, *query* and *queryhit* used to exchange data, and *push* used to deal with peers behind the firewall. A Gnutella message consists of header and content. The attributes of the header are shown as follows:

```
Gnutella message header
+------------+------------+-----+------+----------------+
| message id | descriptor | ttl | hops | payload length |
+------------+------------+-----+------+----------------+
```

The *message id* field is used to detect whether a message already arrived at a certain peer before. The *payload descriptor* field indicates the type of a message such as ping

(0×00), pong (0×01), query (0×80), queryhit (0×81) and push (0×40). The *ttl* field is the number of times that a message can be forwarded in the network while the *hops* field is the number of times that a message has been forwarded. The *payload length* field is the in-byte size of the content that immediately follows the header. The detailed structure of the content depending on message types is defined in the protocol specification [14].

The MapReduce P2P system only uses super peers to perform MapReduce operations. The Gnutella protocol is therefore extended to enable the MapReduce operations on these peers by adding four types of messages: *group*, *join*, *mrinit* and *mrfin*. The *group* and *join* messages work as requests and responses to establish MapReduce groups, whereas the *mrinit* and *mrfin* messages work as requests and responses to handle MapReduce task executions. We refer MapReduce group, MapReduce task and super peer as group, task and peer, respectively, for simplicity.

```
group        join
+----------+ +----------+------+------------+------------+
| group id | | group id | port | ip address | capability |
+----------+ +----------+------+------------+------------+
```

The *group* and *join* messages contain the *group id* field which is the unique identifier of a group in the network. While connecting to the network, a peer maintains connections to the neighboring peers through which the peer can broadcast the *group* messages to the network. Peers accepting to join the group respond with the *join* messages including the same group identifier, *port*, *address* and *capability*. A peer initiating a group becomes the master of the group, while respondents become the slaves or the backup slaves.

```
mrinit                  mrfin
+---------+-----------+ +---------+--------+
| peer id | task list | | peer id | result |
+---------+-----------+ +---------+--------+
```

The *mrinit* and *mrfin* messages contain the *peer id* field which uniquely identifies the master and the slaves in a peer group. The *task list* field presents a set of tuples in a text string: group identifier, slave identifier, input data, task instructions related to MapReduce operations, e.g., a slave retrieves a data chunk from an FTP server and executes the *mapper* operation. When finishing tasks, the slaves send results together with their peer identifier to the master in the *mrfin* messages.

4 Implementation

The implementation of the extended Gnutella protocol includes new messages to support the MapReduce operations, task and failure management. Each peer in this extended Gnutella network is capable of forming groups and executing the MapReduce operations. When a peer wants to solve a distributed problem, it sends the *group* messages to other peers to form a group. Peers receiving the *group* messages can accept to join the group by sending the *join* messages. The peer creating the group works as the master while the other peers of the group work as the slaves or the backup slaves depending on their quick responses. The master manages task assignment and data distribution. The slaves perform the assigned tasks on the retrieved data and send the processed data to the master. The implementation of the MapReduce P2P system focuses on three main modules that contain several components as shown in Fig. 2.

Network module contains a peer controller responsible for dispatching various types of messages to appropriate peers and handlers. The peer controller running on a separate process manages several threads for peer activities. One thread keeps track of the status of all connected peers and notifies if a peer fails to respond. Another thread maintains the number of the neighboring peers or the group members by connecting to a list of preference hosts recorded previously from the P2P network. When performing the MapReduce operations, the other two threads are created to control message exchanges among this controller, the group and MapReduce handlers for group management and task execution.

Group module contains a group handler responsible for managing groups and providing group information to other handlers. When a peer deals with the MapReduce operations, the group handler running on a separate process starts a thread to initiate a group with a unique identifier. Cooperating with the peer controller, this thread requests other peers to join the group by spreading out the *group* messages. When receiving the *join* messages, the thread adds the respondents to the group as the slaves or the backup slaves. The group handler also uses another thread to exchange messages with the MapReduce handler. These messages provide the information of the group for task assignment and slave replacement.

MapReduce module contains MapReduce, task and data handlers which control the MapReduce operations. Figure 3 plots the relationship of the handlers and controller in a UML diagram. Depending on the role of peers, these handlers can operate differently. For the master, the MapReduce handler invokes the task and data handlers to split the input data into fixed-size chunks of 16 MB and upload these chunks to a mediate repository. The MapReduce handler obtains the slaves from the group handler and the data locations from the data handler, and cooperates with the task handler to assign each slave with a task that includes data and operations. The task handler monitors the execution of the slaves by checking task assignment that is sent to the slaves through the *mrinit* messages. When receiving the *mrfin* messages, the handler forwards results to the MapReduce handler for generating further tasks or the final result. For the slaves, the MapReduce handler activates the task handler with task assignment obtained from the *mrinit* message, and invokes the data handler to download data chunks from the mediate repository to

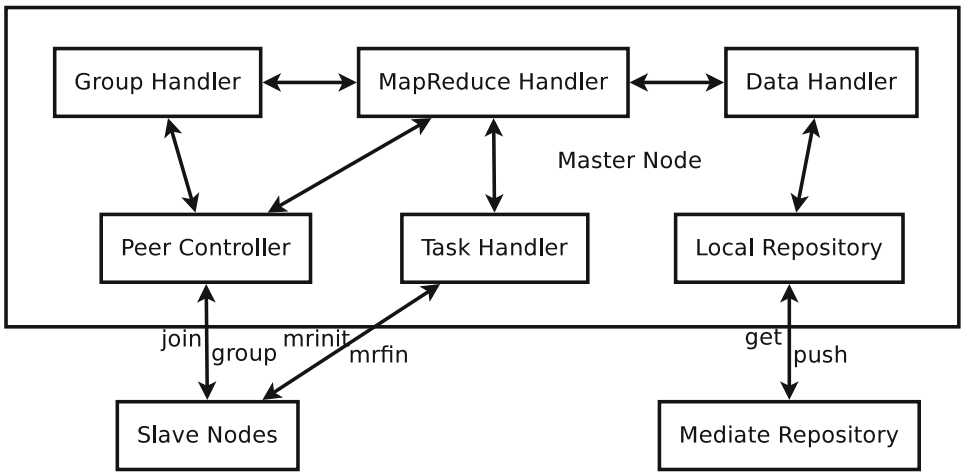

Fig. 2 MapReduce P2P component implementation

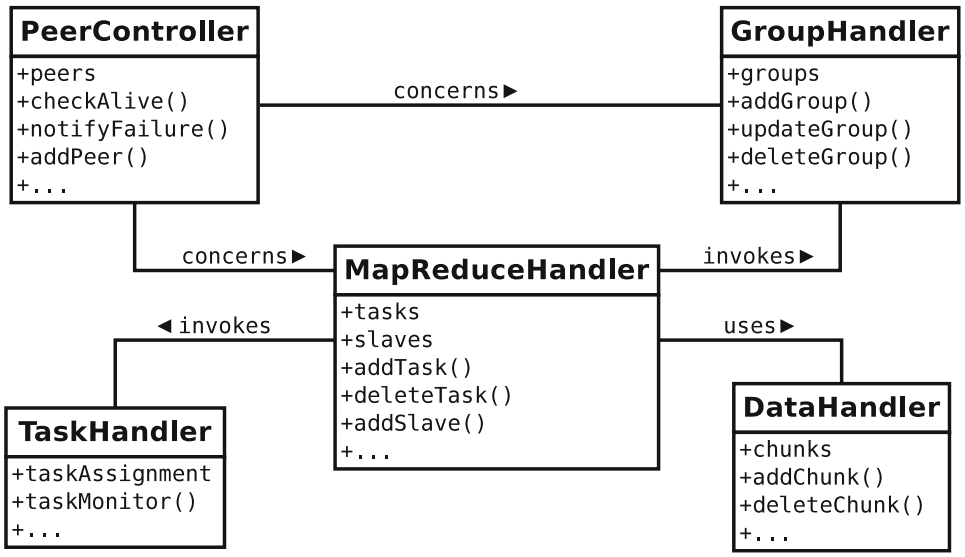

Fig. 3 Peer extension to MapReduce operations represented as a UML diagram

the local repository. Using task assignment, the task handler obtains and processes data chunk, monitors task execution on the data chunk, and sends results to the master through the *mrfin* message.

The Gnutella protocol and modules are implemented by Python. Each peer also contains a MySQL [21] database to store computing data, peer data and messages. In addition, the system supports a web application interface on some peers that allow users to submit distributed computing problems to the system. The web interface is built by Django [12], an open source web application framework written in Python. Django provides several facilities for developing web applications and integrates well into web servers, such as Apache HTTP Servers [2].

4.1 Task management

The slaves need to follow instructions to perform computing tasks. Task assignment descriptions allow users to describe these instructions. A simple assignment instructs the slaves to obtain data chunks from an FTP server, perform specific operations on the data chunks, and return results to the master. A complex assignment requires a workflow language and a directed acyclic graph to present task dependencies and assign tasks, e.g., Condor workflow management system [8] or Pegasus workflow management system [11]. However, we do not need such a complex system for the MapReduce operations because the main instruction of task assignments in our system is to specify operations and data chunks for the master and the slaves. Solving the distributed word count problem is an example. This problem requires to count a number of word occurrences in a text collection using the MapReduce operations: *mapper* and *reducer*. Given a key–value pair of a document identifier and a document, the *mapper* operation performed on the slaves takes the input key–value pair, tokenizes document, and produces intermediate key–value pairs for every word, where word is a key and number of word occurrences is a value. All the intermediate key–value pairs are sorted and hashed into buckets. The key–value pairs with the same key are placed in the same bucket. The *reducer*

operation performed on the master simply sums up all counts associated with each word and then emits the final key–value pairs with word as a key and count as a value.

Algorithm 1: Task Processing

```
Input:    pid: peer identifier
          C: set of tasks {c_i}
Variable: R: set of temporary results {r_j}
          i, j, k, h: integer indexes ≥ 0
Output:   F: set of final results {f_k}
1   F ← ∅, R ← ∅
2   if pid ≠ 0 do                    // for slaves
3     while C ≠ ∅ do
4       r_i ← process(c_i)
5       R ← R ∪ {r_i}
6       C ← C − {c_i}
7     end while
8   else                             // for master
9     C ← create({c_h})
10    while R ≠ ∅ do
11      if finalize(r_j) do
12        f_j ← store(r_j)
13        F ← F ∪ {f_j}
14      else
15        c_j ← process(r_j)
16        C ← C ∪ {c_j}
17      end if
18      R ← R − {r_j}
19    end while
20  end if
21  return F
```

Algorithm 1 presents task processing on the master and the slaves. The algorithm's input and output parameters are the set C of tasks ($\{c_i\}$) created and uploaded by the master, the set R of temporary results ($\{r_j\}$) processed and sent by slaves, and the set F of final results ($\{f_k\}$). Note that i, j, k, h are integer indexes (≥ 0) for sets C, R and F, e.g., c_i is the ith element in C. For implementation, the master uploads data chunks on an FTP server and sends tasks to the slaves that in turn send temporary results to the master using messages. There are four functions: *create* to create a set of tasks, *process* to apply *mapper* or *reducer* for data chunks, *finalize* to check if temporary result needs further processing, and *store* to store temporary result as final result. Slaves ($pid \neq 0$) obtain each task c_i from C, process the task, store the result in R for the master, and then repeat these steps. The master creates and uploads a set of tasks, then joins a loop while until there is no result for processing. In the loop, it checks temporary results sent by slaves to store in F or to create a new tasks in C, then repeats these steps. The master and slaves stop when there is no task generated.

4.2 Failure management

Peers regularly encounter network delay or disconnection, causing the failure of performing computing tasks on the slaves. This situation is rather prevailing in P2P networks where peers can join and leave arbitrarily. Failure management allows each peer group to maintain a list of backup slaves for replacing failed slaves. While establishing a peer group, the master receives several joining messages from peers. Depending on task assignment, some peers become slaves and other peers become backup slaves. Algorithm 2 illustrates the replacement process of failed slaves. The algorithm's input parameters are the set A of active slaves and tasks ($\{(a_i, t_j)\}$), the set B of backup slaves ($\{b_k\}$), and the set F of failed slaves ($\{f_h\}$). Similarly, i, j, k, h are integer indexes (≥ 0) for sets A, B and F. When failed slaves are detected by probing alive messages, the master checks for each failed slave if B is not empty, obtains the task assigned to the failed slave, transfers the task to a new backup slave, and updates A, B and F. The peer group fails if there is not enough backup slaves for the failed slaves.

Algorithm 2: Slave Replacement

```
Input:    A: set of active slaves & tasks {(a_i, t_j)}
          B: set of backup slaves {b_k}
          F: set of failed slaves {f_h}
Variable: i, j, k, h: integer indexes ≥ 0
Output:   done (True: success, False: failure)
1   done ← True
2   while F ≠ ∅ do
3     if B ≠ ∅ then
4       A ← A − {(f_h, t_j)}
5       A ← A ∪ {(b_k, t_j)}
6       B ← B − {b_k}
7       F ← F − {f_h}
8     else
9       done ← False
10    end if
11  end while
12  return done
```

While the failure of the master or slaves causes similar severity on computation problems, the probability of slave failure is much higher than the probability of master failure. Establishing and maintaining the set of backup slaves come at low cost, and the master does not need to send additional messages to backup slaves. However, replacing failed slaves can increase computation cost because some failed tasks are re-computed on the replaced slaves. The failure of the master has been investigated in the other study [19].

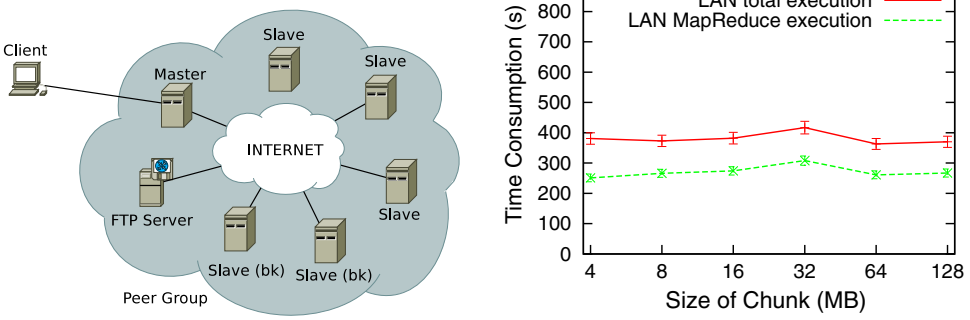

Fig. 4 Establishment of a peer group with failure management for the MapReduce operations (*left*). Performance evaluation of the 8-peer group on LAN using various chunk sizes (*right*)

5 Evaluation

The MapReduce P2P prototyping implementation can be used for solving distributed computing problems. The evaluation of this implementation as a testbed is concerned with several issues including network latency, peer reliability, distributed problems to which the following experiments address. Evaluation configuration includes several workstations and servers:

- 16 workstations: HP Pro Intel (tm) Core i3 Processor 3.30 GHz, 2 GB RAM, 512 GB HDD running at the networking laboratory of our university
- two servers: Sunfire X4200 Dual Core AMD Opteron (tm) Processor 2.6 GHz, 8 GB RAM, 2×150 GB HDD running at the networking laboratory of our university
- 4 workstations: Dell Pentium Dual Core Processor 2.7 GHz, 2 GB RAM, 512 GB HDD running outside our university.

The workstations use Ubuntu 10.04 and Windows 7 Professional, and the servers use Ubuntu Server 10.04. We configured Apache Hadoop [1] on the cluster of laboratory workstations (LAN) and the MapReduce P2P prototyping system on both lab workstations (LAN) and other workstations (INT) as peers. The workstations can form peer groups either on the same local area network for small research groups or on the Internet (INT) for the individual users. Figure 4 on the left side plots the network topology of a peer group for experiments. The peer group maintains a list of backup slaves (bk) used for replacing the failed slaves during computation. The solved problem is the distributed word count problem, and datasets are text files with various sizes ranging from 50 to 400 MB. For each experiment, the master divides a text file into a number of equal sized chunks depending on the selected chunk size, chunks are consecutively assigned to slaves to perform the *mapper* operation, i.e., counting words in a chunk. Note that chunks are uploaded to and downloaded from FTP servers. When all chunks are processed by slaves, the master collects results and performs the *reducer* operation. Assigning *mapper* and *reducer* operations and chunks to slaves are described in task assignment. We perform each experiment several times to collect results and compute mean and standard deviation. All plots contain errorbars that indicate the errors of experiments. The memory usage is measured by the maximum memory usage of peers, e.g., the 2-peer group allocates approximately 200 MB per each peer, while the 8-peer group only allocates approximately 50 MB per each peer for the 400 MB dataset.

The first experiment evaluates the performance of the 8-peer group on LAN using various chunk sizes. The 1 GB dataset is divided into chunks with sizes from 4 to 128 MB. Performance metrics include total execution time measured on the master and average execution time on the slaves because the slaves perform the *mapper* and *reducer* operations concurrently. The group performs similarly with various chunk sizes except for the 32 MB chunk size, as shown in Fig. 4 on the right side. We observe that a large chunk causes high communication time, while a small chunk causes connection problem because slaves must wait for too long to fetch chunks several times from the FTP server. The experiments fail several times with the small chunk sizes as a consequence. This problem becomes more serious for peer groups running on the Internet, thus we consider to choose the 16 MB chunk size for further experiments.

The second experiment compares the performance of the 2-node Hadoop group (2 HDP nodes), the 2-peer group on LAN (2 LAN peers) and the Internet (2 INT peers). Both the master and slave perform the *mapper* operation on chunks, while the master performs the *reducer* operation. Performance metric includes total execution time measured on the master. Figure 5 on the left side shows that the peer groups on LAN and the Internet outperform the Hadoop group that spends 750s to process the 400 MB dataset. The peer group on LAN performs efficiently on all the datasets. While the peer groups use an FTP server, the Hadoop group uses a distributed file system for data distribution that causes high time consumption. The peer group running on the Internet faces performance reduction due to the effect of high communication time.

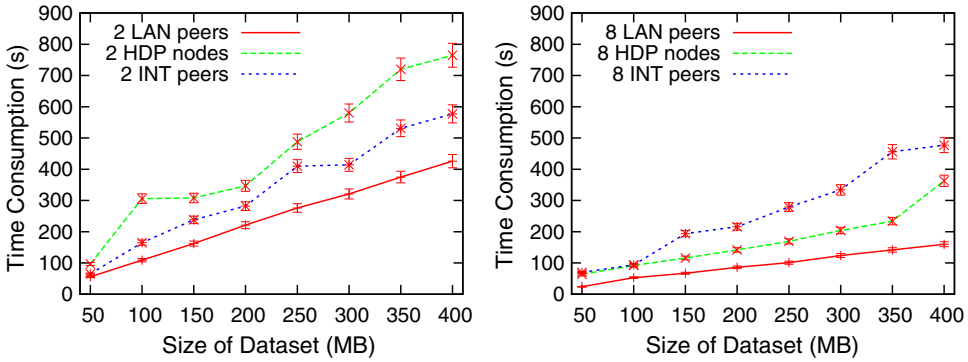

Fig. 5 Performance comparison of the 2-node Hadoop group, the 2-peer groups on LAN and the Internet (*left*). Performance comparison of the 2-node Hadoop group, the 2-peer groups on LAN and the Internet (*right*)

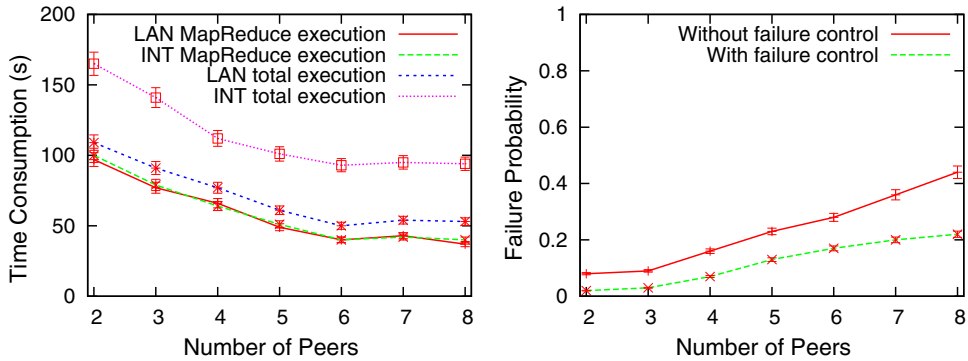

Fig. 6 Performance comparison of the peer groups on LAN and the Internet using the separate FTP servers (*left*). Failure probability of the peer groups on the Internet (*right*)

The third experiment repeats the second experiment using a number of 8 nodes or peers. The 8-peer group on LAN outperforms the other groups on all the datasets, the Hadoop group performs closely the 8-peer group on LAN, and the 8-peer group on the Internet performs poorly, as depicted in Fig. 5 on the right side. The Hadoop group tends to take advantage of a large number of nodes to improve computation time. The peer group on the Internet heavily depends on the communication time of the FTP server, i.e., downloading and uploading data. Computing large datasets, the groups of several Internet peers are also susceptible to operation failures. Failure management thus helps solving this problem, but causes performance decline due to the cost of slave replacement.

The fourth experiment compares the performance of various peer groups on LAN and the Internet with the separate FTP servers. The size of dataset is chosen by 100 MB. This experiment investigates the effect of using the separate FTP servers on the Internet because communication time can be considerably influenced by network delay: the masters need to upload the datasets to the FTP server and the slaves need to download the data chunks from the FTP server. Figure 6 on the left side shows that the average MapReduce operations' execution time is similar for the slaves on both settings. The total execution time for the masters on LAN is averagely 30 % better than that on the Internet. The difference of the total time and MapReduce time is the communication time between the masters, the slaves and the separate FTP servers.

The fifth experiment reports the failure probability of the peer group on the Internet with failure management. The size of dataset is chosen by 400 MB. Figure 6 on the right side specifies that the number of the failed slaves increases as the number of the slaves increases. Without failure management, the 5-peer group possesses a failure probability value of 0.2, and the 8-peer group possesses a failure probability value of 0.4 approximately. Failure probability reduces twice for the same peer groups with failure management. However, even using failure management, the peer groups can still fail due to the failure of several slaves, especially when processing large datasets.

6 Conclusions

We have proposed a MapReduce P2P framework for distributed computing applications. This framework aims to exploit leisure resources including storage, bandwidth and processing power shared among peers on P2P networks to perform MapReduce operations. The framework can be useful for various types of users: researchers cannot afford the expensive clusters for infrequent demands of solving the distributed problems, or Internet users from social and P2P network communities perform experiments on distributed environment. We have recently developed a MapReduce P2P prototyping system that extends the Gnutella protocol to enable group formation, data distribution, and MapReduce operations. This

system also contains failure and task management components that alleviate the severity of peer failure on the Internet and facilitate the applicability of various distributed applications using task management, respectively. The performance evaluation of the system exposes several remarks. With the same number of peers, LAN peer groups outperform Internet peer groups and Hadoop groups. Internet peer groups face performance decline as the number of peers increases due to peer failure and network latency. Communication time of Internet peer groups is approximately 30 % greater than LAN peer groups. The system has not yet been verified with different distributed problems that can possess complicated task assignment and dependency. The system also lacks a component to monitor high workload on peers, which may lead to peer failure.

The MapReduce P2P framework can also be applied to distributed computing applications, especially applications running on P2P networks. These applications process large amount of data on distributed workstations on the Internet. We apply this framework to improve the computation component of DisCaRia, a distributed case-based reasoning (CBR) system for resolving faults in network and communication systems [28–33]. DisCaRia takes advantage of P2P technology to extend the conventional CBR systems [3], thus exploring problem solving knowledge resources in distributed environments, such as expert communities, ticket tracking systems (TTSs), forums and archives. Each peer contains an independent CBR component and exploits knowledge resources in parallel; the system therefore enhances the performance of managing huge datasets on various peers and the quality of various output solutions. The main disadvantage of DisCaRia is high computation cost and low efficiency of the computation component as the size of fault datasets increases. Note that fault reports contain several symptoms, error messages, distinct keywords, etc. MapReduce operations can deal with this problem by processing a large number of fault reports on various peers quickly and efficiently.

Acknowledgments This research work is funded by Vietnam National Foundation for Science and Technology Development (NAFOSTED) under grant number 102.02- 2011.01.

References

1. Apache Hadoop Project. http://hadoop.apache.org/. Accessed Mar 2013
2. Apache Server Project. http://httpd.apache.org/. Accessed Mar 2013
3. Aamodt, A., Plaza, E.: Case-based reasoning: foundational issues, methodological variations, and system approaches. AI Commun. **7**(1), 39–59 (1994)
4. Berkovsky, S., Kuflik, T., Ricci, F.: P2P case retrieval with an unspecified ontology. In: Proceedingd of the 6th International Conference on Case-Based Reasoning, pp. 91–105. Springer, Berlin (2005)
5. Camarillo, G.: Peer-to-Peer (P2P) Architecture: Definition, Taxonomies, Examples, and Applicability (2009)
6. Clarke, I., Sandberg, O., Wiley, B., Hong, T.W.: Freenet: a distributed anonymous information storage and retrieval system. In: Proceedings of the International Workshop on Design Issues in Anonymity and Unobservability, pp. 46–66. Springer, Heidelberg (2000)
7. Cohen, B.: Incentives build robustness in bitorrent. In: Proceedings of the 1st Workshop on Economics of Peer-to-Peer Systems (2003)
8. Couvares, P., Kosar, T., Roy, A., Weber, J., Wenger, K.: Workflow management in condor. In: Taylor, I., Deelman, E., Gannon, D., Shields, M. (eds.) Workflows for e-Science, pp. 357–375. Springer, London (2007)
9. Dang, H.T., Tran, H.M., Vu, P.N., Nguyen, A.T.: Applying MapReduce framework to peer-to-peer computing applications. In: Proceedings of the 4th International Conference on Computational Collective Intelligence Technologies and Applications (ICCCI 2012), pp. 69–78. Springer, Berlin (2012)
10. Dean, J., Ghemawat, S.: Mapreduce: simplified data processing on large clusters. Commun. ACM **51**, 107–113 (2008)
11. Deelman, E., Singh, G., Su, M., Blythe, J., Gil, Y., Kesselman, C., Mehta, G., Vahi, K., Berriman, G., Good, J., Laity, A., Jacob, J., Katz, D.: Pegasus: a framework for mapping complex scientific workflows onto distributed systems. Sci. Program. **13**(3), 219–237 (2005)
12. Django Project. http://www.djangoproject.com/. Accessed Mar 2013
13. Faroo Project. http://www.faroo.com/. Accessed June 2012
14. Gnutella Protocol Specification version 0.4. http://rfc-gnutella.sourceforge.net/rfc-gnutella.zip (2001). Accessed June 2012
15. Heckmann, O., Bock, A., Mauthe, A., Steinmetz, R.: The eDonkey file sharing network. Proc. GI Jahrestagung **2**, 224–228 (2004)
16. Kazaa Software. http://www.kazaa.com/. Accessed June 2012
17. Luu, T., Klemm, F., Podnar, I., Rajman, M., Aberer, K.: Alvis peers: a scalable full-text peer-to-peer retrieval engine. In: Proceedings of the International Workshop on Information Retrieval in Peer-to-Peer Networks (P2PIR '06), pp. 41–48. ACM, New York (2006)
18. Marozzo, F., Talia, D., Trunfio, P.: Adapting mapreduce for dynamic environments using a peer-to-peer model. In: Proceedings of the 1st Workshop on Cloud Computing and its Applications (CCA '08), Chicago, USA (2008)
19. Marozzo, F., Talia, D., Trunfio, P.: A framework for managing mapreduce applications in dynamic distributed environments. In: Proceedings of the 19th International Euromicro Conference on Parallel, Distributed and Network-Based Processing, pp. 149–158. IEEE Computer Society, Los Alamitos (2011)
20. Maymounkov, P., Mazières, D.: Kademlia: a peer-to-peer information system based on the XOR Metric. In: Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS '01), pp. 53–65. Springer, London (2002)
21. MySQL Database Software. http://www.mysql.com/. Accessed Mar 2013
22. Napster Project. http://www.napster.com/. Accessed June 2012
23. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Schenker, S.: A scalable content addressable network. In: Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM '01), pp. 161–172. ACM Press, New York (2001)

24. Rowstron, A., Druschel, P.: Pastry: scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In: Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms (Middleware '01), vol. 2218, pp. 329–351. Springer, London (2001)
25. Schlosser, M., Sintek, M., Decker, S., Nejdl, W.: A scalable and ontology-based P2P infrastructure for semantic web services. In: Proceedings of the 2nd International Conference on Peer-to-Peer Computing (P2P '02), p. 104. IEEE Computer Society, Washington, DC (2002)
26. Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., Balakrishnan, H.: Chord: a scalable peer-to-peer lookup service for internet applications. In: Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM '01), pp. 149–160. ACM Press, New York (2001)
27. Tatarinov, I., Ives, Z., Madhavan, J., Halevy, A., Suciu, D., Dalvi, N., Dong, X., Kadiyska, Y., Miklau, G., Mork, P.: The piazza peer data management project. SIGMOD Rec. **32**(3), 47–52 (2003)
28. Tran, H.M., Chulkov, G., Schönwälder, J.: Crawling bug tracker for semantic bug search. In: Proceedings of the 19th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM '08), pp. 55–66. Springer, Berlin (2008)
29. Tran, H.M., Schönwälder, J.: Distributed case-based reasoning for fault management. In: Proceedings of the 1st International Conference on Autonomous Infrastructure, Management and Security, pp. 200–203. Springer, Berlin (2007)
30. Tran, H.M., Schönwälder, J.: Fault representation in case-based reasoning. In: Proceedings of the 18th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, pp. 50–61. Springer, Berlin (2007)
31. Tran, H.M., Schönwälder, J.: Heuristic search using a feedback scheme in unstructured peer-to-peer networks. In: Proceedings of the 5th International Workshop on Databases, Information Systems and Peer-to-Peer Computing. Springer, Berlin (2007)
32. Tran, H.M., Schönwälder, J.: Fault resolution in case-based reasoning. In: Proceedings of the 10th Pacific Rim International Conference on Artificial Intelligence (PRICAI '08), pp. 417–429. Springer, Berlin (2008)
33. Tran, H.M., Schönwälder, J.: Evaluation of the distributed case-based reasoning system on a distributed computing platform. In: Proceedings of the 7th International Symposium on Frontiers of Information Systems and Network Applications (FINA 2011), pp. 53–58 (2011)
34. Yacy Project. http://www.yacy.de/. Accessed June 2012
35. Yang, B., Garcia-Molina, H.: Designing a super-peer network. In: Proceedings of the 19th International Conference on Data Engineering (ICDE'03), p. 49, IEEE Computer Society, Los Alamitos (2003)
36. Zhao, B., Huang, L., Stribling, J., Rhea, S., Joseph, A., Kubiatowicz, J.: Tapestry: a resilient global-scale overlay for service deployment. IEEE J. Sel. Areas Commun. **22**(1), 41–53 (2003)